

A Formal Specification of Dynamic Protocols for Open Agent Systems

Alexander Artikis

Abstract Multi-agent systems where the agents are developed by parties with competing interests, and where there is no access to an agent’s internal state, are often classified as ‘open’. The member agents of such systems may inadvertently fail to, or even deliberately choose not to, conform to the system specification. Consequently, it is necessary to specify the normative relations that may exist between the agents, such as permission, obligation, and institutional power. The specification of open agent systems of this sort is largely seen as a design-time activity. Moreover, there is no support for run-time specification modification. Due to environmental, social, or other conditions, however, it is often required to revise the specification during the system execution. To address this requirement, we present an infrastructure for ‘dynamic’ specifications, that is, specifications that may be modified at run-time by the agents. The infrastructure consists of well-defined procedures for proposing a modification of the ‘rules of the game’, as well as decision-making over and enactment of proposed modifications. We evaluate proposals for rule modification by modelling a dynamic specification as a metric space, and by considering the effects of accepting a proposal on system utility. Furthermore, we constrain the enactment of proposals that do not meet the evaluation criteria. We employ the action language *C+* to formalise dynamic specifications, and the ‘Causal Calculator’ implementation of *C+* to execute the specifications. We illustrate our infrastructure by presenting a dynamic specification of a resource-sharing protocol.

1 Introduction

A particular kind of Multi-Agent System (MAS) is one where the member agents are developed by different parties that have conflicting goals, and where there is no access to an agent’s internal state. A key characteristic of this kind of MAS, due to the globally inconsistent goals of its members, is the high probability of non-conformance to

Institute of Informatics & Telecommunications,
National Centre for Scientific Research “Demokritos”,
Athens 15310, Greece
E-mail: a.artikis@iit.demokritos.gr, a.artikis@acm.org

the specifications that govern the members' interactions. A few examples of this type of MAS are electronic marketplaces, virtual organisations, and digital media rights management applications. MAS of this type are often classified as 'open'.

Open MAS can be viewed as instances of *normative systems* [46]. A feature of this type of system is that actuality, what is the case, and ideality, what ought to be the case, do not necessarily coincide. Therefore, it is essential to specify what is permitted, prohibited, and obligatory, and perhaps other more complex normative relations that may exist between the agents. Among these relations, considerable emphasis has been placed on the representation of *institutional power* [47]. This is a standard feature of any normative system whereby designated agents, when acting in specified roles, are empowered by an institution to create specific relations or states of affairs. Consider, for example, the case in which an agent is empowered by an institution to award a contract and thereby create a bundle of normative relations between the contracting parties.

Several approaches have been proposed in the literature for the specification of open MAS. The majority of these approaches offer 'static' specifications, that is, there is no support for run-time specification modification. In some open MAS, however, environmental, social or other conditions may favour, or even require, specifications that are modifiable during the system execution. Consider, for instance, the case of a malfunction of a large number of sensors in a sensor network, or the case of manipulation of a voting procedure due to strategic voting, or when an organisation conducts its business in an inefficient manner. Therefore, we present in this paper an infrastructure for 'dynamic' specifications, that is, specifications that are developed at design-time but may be modified at run-time by the members of a system. The presented infrastructure is an extension of our work on static specifications [5, 7], and is motivated by 'dynamic argument systems' [11]. These are argument systems in which, at any point in the disputation, agents may start a meta level debate, that is, the rules of order become the current point of discussion, with the intention of altering these rules.

Our infrastructure for dynamic specifications allows agents to alter the specification of a protocol P during the protocol execution. P is considered an 'object' protocol; at any point in time during the execution of the object protocol the participants may start a 'meta' protocol in order to decide whether the object protocol specification should be modified. Moreover, the participants of the meta protocol may initiate a meta-meta protocol to decide whether to modify the specification of the meta protocol, or they may initiate a meta-meta-meta protocol to modify the specification of the meta-meta protocol, and so on.

Unlike existing approaches on dynamic specifications, we place emphasis on the procedure with which agents initiate a meta protocol. We distinguish between successful and unsuccessful attempts to initiate a meta protocol by identifying the conditions in which an agent has the institutional power to propose a specification change. We evaluate an agent's proposal for specification change by modelling a dynamic specification as a *metric space* [13], and by taking into consideration the effects of accepting a proposal on system utility. We constrain the enactment of proposals that do not meet the evaluation criteria. Furthermore, we formalise procedures for role-assignment in a meta level, that is, we specify which agents may participate in a meta protocol, and the roles they may occupy in the meta protocol.

We employ a resource-sharing protocol to illustrate our infrastructure for dynamic specifications: the object protocol concerns resource-sharing while the meta protocols are voting protocols. In other words, at any time during a resource-sharing procedure

the agents may vote to change the rules that govern the management of resources. The resource-sharing protocol was chosen for the sake of providing a concrete example. In general, the object protocol may be any protocol for open MAS, such as a protocol for coordination or e-commerce. Similarly, a meta protocol can be any procedure for decision-making over specification modification (argumentation, negotiation, and so on).

We encode dynamic MAS specifications in executable action languages. In this paper we employ the action language $C+$ [35], a formalism with explicit transition system semantics. The $C+$ language, when used with its associated software implementation, the ‘Causal Calculator’ (CCALC), supports a wide range of computational tasks of the kind that we wish to perform on MAS specifications.

The remainder of this paper is structured as follows. First, we present the action language $C+$. Second, we review a static specification of a resource-sharing protocol, and show how CCALC can be used to prove properties of the static specification. Third, we present a dynamic specification of the resource-sharing protocol and an infrastructure for modifying the protocol specification during the protocol execution. We then show how CCALC can be used to prove properties of the infrastructure for dynamic specifications, as well as support run-time activities by computing the normative relations current at each time. Finally, we summarise our work, discuss related research, and outline directions for further work.

2 The $C+$ Language

$C+$, as mentioned above, is an action language with an explicit transition system semantics. We describe here the version of $C+$ presented in [35].

2.1 Basic Definitions

A *multi-valued propositional signature* is a set σ of symbols called *constants*, and for each constant $c \in \sigma$, a non-empty finite set $dom(c)$ of symbols, disjoint from σ , called the *domain* of c . For simplicity, in this presentation we will assume that every domain contains at least two elements.

An *atom* of signature σ is an expression of the form $c=u$ where $c \in \sigma$ and $u \in dom(c)$. A Boolean constant is one whose domain is the set of truth values $\{t, f\}$. When c is a Boolean constant we often write c for $c=t$ and $\neg c$ for $c=f$. A *formula* φ of signature σ is any propositional combination of atoms of σ . An *interpretation* I of σ is a function that maps every constant in σ to an element of its domain. An interpretation I *satisfies* an atom $c=u$ if $I(c)=u$. The satisfaction relation is extended from atoms to formulas according to the standard truth tables for the propositional connectives. A *model* of a set X of formulas of signature σ is an interpretation of σ that satisfies all formulas in X . If every model of a set X of formulas satisfies a formula φ then X *entails* φ , written $X \models \varphi$.

2.2 Syntax

The representation of an action domain in $C+$ consists of *fluent* constants and *action* constants.

- Fluent constants are symbols characterising a state. They are divided into two categories: simple fluent constants and statically determined fluent constants. Simple fluent constants are related to actions by *dynamic laws*, that is, laws describing a transition $(s_i, \varepsilon_i, s_{i+1})$ from a state s_i to its successor state s_{i+1} . Statically determined fluent constants are characterised by *static laws*, that is, laws describing an individual state, relating them to other fluent constants. Static laws can also be used to express constraints between simple fluent constants. Static and dynamic laws are defined below.
- Action constants are symbols characterising state transitions. In a transition $(s_i, \varepsilon_i, s_{i+1})$, the transition label ε_i , also called an ‘event’, represents the actions performed concurrently by one or more agents or occurring in the environment. Transitions may be non-deterministic. Action constants are used to name actions, attributes of actions, or properties of transitions as a whole.

An *action signature* (σ^f, σ^a) is a non-empty set σ^f of fluent constants and a non-empty set σ^a of action constants. An *action description* D in $C+$ is a non-empty set of *causal laws* that define a transition system of a particular type. A causal law can be either a *static law* or a *dynamic law*. A static law is an expression

$$\text{caused } F \text{ if } G \quad (1)$$

where F and G are formulas of fluent constants. In a static law, constants in F and G are evaluated on the same state. A dynamic law is an expression

$$\text{caused } F \text{ if } G \text{ after } H \quad (2)$$

where F , G and H are formulas such that every constant occurring in F is a simple fluent constant, every constant occurring in G is a fluent constant, and H is any combination of fluent constants and action constants. In a transition from state s_i to state s_{i+1} , constants in F and in G are evaluated on s_{i+1} , fluent constants in H are evaluated on s_i and action constants in H are evaluated on the transition itself. F is called the *head* of the static law (1) and the dynamic law (2).

The full $C+$ language also provides *action dynamic laws*, which are expressions of the form

$$\text{caused } \alpha \text{ if } H$$

where α is a formula containing action constants only and H is a formula of action and fluent constants. We will not use action dynamic laws in this paper and so omit the details in the interests of brevity.

The $C+$ language provides various abbreviations for common forms of causal law. For example, a dynamic law of the form

$$\text{caused } F \text{ if } \top \text{ after } H \wedge \alpha$$

where α is a formula of action constants is often abbreviated as

$$\alpha \text{ causes } F \text{ if } H$$

In the case where H is \top the above is usually written as $\alpha \text{ causes } F$.

When presenting the resource-sharing protocol specification, we will often employ the **causes** abbreviation to express the effects of the agents' actions. We will also employ the $C+$ abbreviation

$$\text{default } F$$

which is shorthand for the static law

$$\text{caused } F \text{ if } F$$

expressing that F holds in the absence of information to the contrary.

When it aids readability, we will write

$$\text{caused } F \text{ iff } G$$

as a shorthand for the pair of static laws

$$\text{caused } F \text{ if } G$$

and

$$\text{default } \neg F$$

Finally, we will express the inertia of a fluent constant c over time as:

$$\text{inertial } c$$

This is an abbreviation for the *set* of dynamic laws of the form (for all values $u \in \text{dom}(c)$):

$$\text{caused } c = u \text{ if } c = u \text{ after } c = u$$

A $C+$ action description is a non-empty set of causal laws. Of particular interest is the sub-class of *definite* action descriptions. A $C+$ action description D is *definite* if:

- the head of every causal law of D is an atom or \perp , and
- no atom is the head of infinitely many causal laws of D .

The $C+$ action description in this paper will be definite.

2.3 Semantics

It is not possible in the space available here to give a full account of the $C+$ language and its semantics. We trust that the $C+$ language, and especially its abbreviations, are sufficiently natural that readers can follow the presentation of the case study in later sections. Interested readers are referred to [35,36] for further technical details. For completeness, we summarise here the semantics of *definite* action descriptions ignoring, as we are, the presence of action dynamic laws (and assuming that the domain of every constant contains at least two elements). We emphasise the transition system semantics, as in [68].

Every action description D of $C+$ defines a labelled transition system, as follows:

- States of the transition system are interpretations of the fluent constants σ^f . It is convenient to identify a state s with the set of fluent atoms satisfied by s (in other words, $s \models f = v$ if and only if $f = v \in s$ for every fluent constant f).

Let $T_{\text{static}}(s)$ denote the heads of all static laws in D whose conditions are satisfied by s :

$$T_{\text{static}}(s) =_{\text{def}} \{F \mid \text{static law (1) is in } D, s \models G\}$$

For a definite action description D , an interpretation s of σ^f is a *state of the transition system defined by D* , or simply, a *state of D* , when

$$s = T_{\text{static}}(s) \cup \text{Simple}(s)$$

where $\text{Simple}(s)$ denotes the set of simple fluent atoms satisfied by s . (So $s - \text{Simple}(s)$ is the set of statically determined fluent atoms satisfied by s .)

- Transition labels of the transition system defined by D are the interpretations of the action constants σ^a .

A *transition* is a triple (s, ε, s') in which s is the initial state, s' is the resulting state, and ε is the transition label. Since transition labels are interpretations of σ^a , it is meaningful to say that a transition label ε satisfies a formula α of σ^a : when $\varepsilon \models \alpha$ we sometimes say that the transition (s, ε, s') is of type α .

- Let $E(s, \varepsilon, s')$ denote the heads of all dynamic laws of D whose conditions are satisfied by the transition (s, ε, s') :

$$E(s, \varepsilon, s') =_{\text{def}} \{F \mid \text{dynamic law (2) is in } D, s' \models G, s \cup \varepsilon \models H\}$$

For a definite action description D , (s, ε, s') is a *transition of D* , or in full, a *transition of the transition system defined by D* , when s and s' are interpretations (sets of atoms) of σ^f and ε is an interpretation of σ^a such that:

- $s = T_{\text{static}}(s) \cup \text{Simple}(s)$ (s is a state of D)
- $s' = T_{\text{static}}(s') \cup E(s, \varepsilon, s')$

For any non-negative integer m , a *path* or *history* of D of length m is a sequence

$$s_0 \varepsilon_0 s_1 \dots s_{m-1} \varepsilon_{m-1} s_m$$

where $(s_0, \varepsilon_0, s_1), \dots, (s_{m-1}, \varepsilon_{m-1}, s_m)$ are transitions of D .

2.4 The Causal Calculator

The Causal Calculator¹ (CCALC) is a software implementation developed by the Action Group of the University of Texas for representing action and change in the $C+$ language, and performing a range of computational tasks on the resulting formalisations. The functionality of CCALC includes computation of ‘prediction’ (temporal projection) and planning queries. Action descriptions in $C+$ are translated by CCALC first into the language of *causal theories* [35] and then into propositional logic. The (ordinary, classical) models of the propositional theory correspond to paths in the transition system described by the original action description in $C+$. To compute an answer to a query, CCALC invokes a satisfiability (SAT) solver to find models of the propositional theory

¹ <http://userweb.cs.utexas.edu/users/tag/cc/>

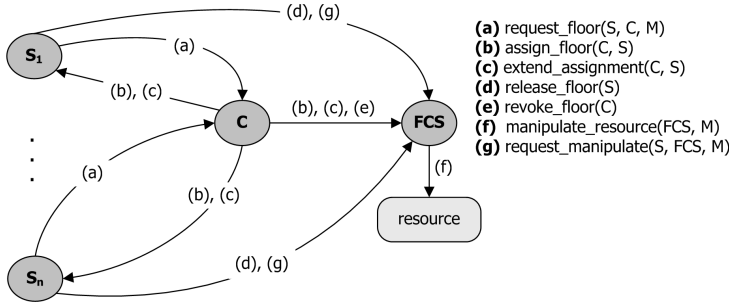


Fig. 1 A Chaired Floor Control Protocol.

which also satisfy the query. A detailed account of CCALC's operation and functionality may be found in [35].

In the following sections we present a $C+$ action description, D^{RS} , expressing a specification of a resource-sharing protocol. More precisely, first we present a static specification of a resource-sharing protocol, and then we present an infrastructure for dynamic specification of a resource-sharing protocol. Moreover, we show how we use CCALC to execute a protocol specification.

3 A Static Resource-Sharing Protocol

We present a specification of a resource-sharing or *floor control* protocol in the style of [7]. In the field of Computer-Supported Co-operative Work the term 'floor control' denotes a service guaranteeing that at any given moment only a designated set of users (subjects) may simultaneously work on the same objects (shared resources), thus creating a temporary exclusivity for access on such resources. We present a 'chair-designated' Floor Control Protocol, that is, a distinguished participant is the arbiter over the usage of a specific resource. For simplicity we assume a single resource.

The protocol roles are summarised below:

- *Floor Control Server (FCS)*, the role of the only participant physically manipulating the shared resource.
- *Subject (S)*, the role of designated participants requesting the floor from the chair, releasing the floor, and requesting from the FCS to manipulate the resource.
- *Chair (C)*, the role of the participant assigning the floor for a particular time period to a subject, extending the time allocated for the floor, and revoking the floor from the subject holding it.

The floor can be either 'granted', denoting that a subject has exclusive access to the resource (by the chair), or 'free', denoting that no subject currently holds the floor. In both cases the floor may or may not be requested by a subject (for example, the floor may be granted to subject S' and requested by subject S'' at the same time).

Figure 1 provides an informal description of the possible interactions between the agents occupying the protocol roles. More details about these interactions will be given presently.

Table 1 shows a subset of the action signature of D^{RS} , that is, the $C+$ the action description expressing the specification of the resource-sharing protocol. Variables start

Table 1 A Subset of the Action Signature of D^{RS} (Part A).

Variable:	Domain:
M	a set of resource manipulation types
Ag, S, S', C	a set of agent ids
Simple Fluent Constant:	Domain:
$role_of(Ag)$	$\{subject, chair, fcs\}$
$holder(S), sanctioned(Ag)$	Boolean
$requested(S)$	a set of resource manipulation types
$best_candidate$	a set of agent ids
$c_alloc(S)$	\mathbb{Z}^+
Statically Determined Fluent Constant (Boolean):	
$powRequest(S, C), powAssign(C, S), powRequestMpt(S, FCS, M)$	
Action Constant σ^{act} (Boolean):	
$request_floor(S, C, M), assign_floor(C, S), request_manipulate(S, FCS, M)$	

with an upper-case letter, and fluent and action constants start with a lower-case letter. The intended reading of the constants of the action signature will be explained below.

It has been argued [47] that the specifications of protocols for open MAS should explicitly represent the concept of institutional power (or, for short, ‘power’), that is, the characteristic feature of institutions whereby designated agents, often when acting in specific roles, are empowered, by the institution, to create or modify facts of special significance in that institution — *institutional facts* — usually by performing a specified kind of act. Searle [66], for example, has distinguished between *brute facts* and institutional facts. Being in physical possession of an object is an example of a brute fact (it can be observed); being the owner of that object is an institutional fact. The resource-sharing protocol specification explicitly represents the concept of institutional power. Moreover, we follow the standard, long-established distinction between institutional power, physical capability, permission and obligation (see [52] for illustrations of this distinction).

According to the resource-sharing protocol specification, all actions are physically possible at any time. In other examples the specification of physical capability could be different.

In this example, a subject S is empowered to request the floor from the chair C when S has no pending requests:

$$\begin{aligned}
 &\text{caused } powRequest(S, C) \text{ iff} \\
 &\quad role_of(S) = subject, \\
 &\quad role_of(C) = chair, \\
 &\quad requested(S) = null
 \end{aligned} \tag{3}$$

$C+$ abbreviations, including iff, were presented in Section 2. The $powRequest$ fluent constant expresses the institutional power to request the floor, while the $role_of$ fluent constant expresses the role an agent occupies. The $requested$ fluent constant records an agent’s requests for the floor — $requested(S) = null$ denotes that S has no requests for the floor. $powRequest$ is a statically determined fluent constant while $role_of$ and $requested$ are simple fluent constants (see Table 1).

Each simple fluent constant of D^{RS} is inertial, that is to say, its value persists by default from one state to the next. The constraint that a fluent constant f is inertial

is expressed in $C+$ by means of the causal law abbreviation:

$$\text{inertial } f \quad (4)$$

Having specified the institutional power to request the floor, it is now possible to define the effects of this action: a request for the floor is eligible to be serviced if and only if it is issued by an agent with the institutional power to request the floor. Requests for the floor issued by agents without the necessary institutional power are ignored. Due to space limitations, we do not present here the $C+$ laws expressing the effects of protocol actions (we show only one such law below).

We chose to specify that a subject is always permitted to exercise its power to request the floor. Moreover, a subject S is permitted to request the floor even if S is not empowered to do so. In the latter case a request for the floor will be ignored by the chair (since S was not empowered to request the floor) but S will not be *sanctioned* since it was not forbidden to issue the request. In general, an agent is sanctioned when performing a forbidden action or not complying with an obligation. A few examples of sanctions will be shown presently (a more thorough treatment of sanctions may be found in [5, 7]). Finally, a subject is never obliged to request the floor.

The chair's power to assign the floor is defined as follows:

$$\begin{aligned} \text{caused } \text{powAssign}(C, S) \text{ iff} \\ \text{role_of}(C) = \text{chair}, \\ \forall S' \neg \text{holder}(S'), \\ \text{best_candidate} = S \end{aligned} \quad (5)$$

The chair C is empowered to assign the floor to S if the floor is free, and S is the best candidate for the floor. The simple fluent constant *holder* expresses whether an agent has been allocated the floor. The simple fluent constant *best_candidate* denotes the best candidate for the floor. The definition of this constant is application-specific. For instance, the best candidate could be the one with the earliest request, that with the most 'urgent' request (however 'urgent' may be defined), and so on.

The result of exercising the power to assign the floor to S is that the floor becomes granted to S for a specified time period. Sending an *assign_floor* message to an agent S without the power to assign the floor to S has no effect on the access rights of S .

In this example, the conditions in which the chair is permitted to assign the floor are expressed as follows:

$$\begin{aligned} \text{caused } \text{perAssign}(C, S) \text{ iff} \\ \text{role_of}(C) = \text{chair}, \\ \forall S' \neg \text{holder}(S'), \\ \text{best_candidate} = S, \\ c_alloc(S) < 3 \end{aligned} \quad (6)$$

The chair is permitted to assign the floor to S if the floor is free, S is the best candidate for the floor, and S has not been allocated the floor the last 3 (or more) times. A simple fluent constant $c_alloc(S)$ is incremented by 1 when S is assigned the floor, and set to 0 when the floor is assigned to some other subject S' .

Note that, according to rules (5) and (6), the chair is not always permitted to exercise its power to assign the floor.

Table 2 A Resource-Sharing Protocol Specification.

Action	Power	Permission	Obligation
$request_floor(S, C, M)$	$requested(S) = null$	\top	\perp
$assign_floor(C, S)$	$\forall S' \neg holder(S'),$ $best_candidate = S$	$\forall S' \neg holder(S'),$ $best_candidate = S,$ $c_alloc(S) < 3$	$\forall S' \neg holder(S'),$ $best_candidate = S,$ $c_alloc(S) < 3$
$request_manipulate(S, FCS, M)$	$holder(S)$	$holder(S)$	\perp

As mentioned above, an agent is subject to penalty when performing a forbidden action. In this case, a chair is subject to penalty when assigning the floor while being forbidden to do so. We record sanctions as follows:

$$\begin{aligned}
& assign_floor(C, S) \text{ causes } sanctioned(C) \text{ if} \\
& \quad role_of(C) = chair, \\
& \quad \neg perAssign(C, S)
\end{aligned} \tag{7}$$

sanctioned is a simple fluent constant. The actual penalty associated with the violation of a prohibition, or non-compliance with an obligation, may come in different flavours. We will show a type of penalty in a later section.

In this example, the conditions in which the chair is obliged to assign the floor are the same as the conditions in which the chair is permitted to assign the floor.

Similarly we specify the power, permission and obligation to perform the remaining protocol actions, and the effects of these actions. For instance, a subject's power to request a manipulation of the shared resource is defined as follows:

$$\begin{aligned}
& caused \ powRequestMpt(S, FCS, M) \text{ iff} \\
& \quad role_of(FCS) = fcs, \\
& \quad holder(S)
\end{aligned} \tag{8}$$

The *powRequestMpt* fluent constant expresses the institutional power to request a resource manipulation. A subject S is empowered to request a resource manipulation of type M from the floor control server FCS if S is the holder of the resource.

The specification of the power, permission or obligation to request a resource manipulation, assign the floor, or perform some other protocol action, should include a deadline stating the time by which the action of requesting a resource manipulation, assigning the floor, etc, should be performed. Including deadlines in the formalisation lengthens the presentation and is omitted here for simplicity. Example formalisations of deadlines may be found in [7].

To summarise, Table 2 presents the conditions in which a protocol participant has the institutional power, permission and obligation to perform an action. To save space, in Table 2 we show only three protocol actions: *request_floor*, *assign_floor* and *request_manipulate*. Moreover, we do not display the *role_of* fluent constant and assume that S denotes an agent occupying the role of subject, C denotes an agent occupying the role of chair, and FCS denotes an agent occupying the role of floor control server.

4 Proving Properties of the Static Resource-Sharing Protocol

The explicit transition system semantics of the $C+$ language enables us to prove various properties of the presented specification, which is expressed by means of the action description D^{RS} . Consider the following example.

Property 4.1 *There is no protocol state in which the floor is free and the chair is not empowered to assign it to the best candidate.*

Proof Assume a state s of the transition system defined by D^{RS} in which the best candidate for the floor is subject S , the floor is free, and the chair C is not empowered to assign the floor to S . In other words, for any S, C ,

$$s \models \text{best_candidate} = S \wedge \forall S' \neg \text{holder}(S') \wedge \text{role_of}(C) = \text{chair} \wedge \neg \text{powAssign}(C, S)$$

Since s is a state of D^{RS} , it is an interpretation of σ^f such that $s = T_{\text{static}}(s) \cup \text{Simple}(s)$, where $T_{\text{static}}(s) =_{\text{def}} \{F \mid \text{static law 'caused } F \text{ if } G' \text{ is in } D^{RS}, s \models G\}$ and $\text{Simple}(s)$ denotes the set of simple fluent atoms satisfied by s (see Section 2.3). From rule (5), and the fact that

$$s \models \text{best_candidate} = S \wedge \forall S' \neg \text{holder}(S') \wedge \text{role_of}(C) = \text{chair}$$

we have that $\text{powAssign}(C, S) \in T_{\text{static}}(s)$. According to our initial assumption, however, in s the chair C is not empowered to assign the floor to S , that is, $\text{powAssign}(C, S) \notin s$, which implies that $s \neq T_{\text{static}}(s) \cup \text{Simple}(s)$. Therefore, s is not a state of D^{RS} . \square

CCALC provides an automated means for proving properties (such as Property 4.1) of a protocol specification formalised in $C+$. We express the $C+$ action description D^{RS} in CCALC's input language and then query CCALC about D^{RS} in order to prove properties of the protocol specification. (Details about the types of query that CCALC computes, and CCALC's input language, may be found in [1, 35, 50].) Consider the following example.

Property 4.2 *A chair is always sanctioned when it performs a forbidden assignment of the floor.*

We instruct CCALC to compute all states s' such that

- (s, ε, s') is a transition of D^{RS} ,
- $s \models \neg \text{perAssign}(C, S) \wedge \text{role_of}(C) = \text{chair}$, and
- $\varepsilon \models \text{assign_floor}(C, S)$.

For every state s' computed by CCALC we obtain

$$s' \models \text{sanctioned}(C)$$

This is due to rule (7).

Note that for some states s' computed by CCALC we obtain $s' \models \text{holder}(S)$, meaning that, in these cases, the chair C was empowered, although forbidden, to assign the floor to subject S (see, respectively, rules (5) and (6) for the specification of the power and permission to assign the floor). CCALC also computed states s' such that

$s' \models \neg \text{holder}(S)$, that is, in these cases, the chair was not empowered to assign the floor to S .

We may prove further properties of the resource-sharing protocol specification, in the manner shown above, such as that an agent is permitted to perform at least one action in every protocol state, an agent is never forbidden and obliged to perform an action, non-compliance with an obligation always leads to a sanction, and so on. Further examples of proving properties of protocol specifications formalised in $C+$ will be presented in Section 6.

5 An Infrastructure for A Dynamic Resource-Sharing Protocol

Being motivated by Brewka [11], we present an infrastructure that allows agents to modify (a subset of) the rules of a protocol at run-time. Regarding our running example, we consider the resource-sharing protocol as an ‘object’ protocol; at any point in time during the execution of the object protocol the participants may start a ‘meta’ protocol in order to potentially modify the object protocol rules — for instance, replace an existing rule-set with a new one. The meta protocol may be any protocol for decision-making over rule modification. For the sake of presenting a concrete example, we chose a voting procedure as a meta protocol, that is, the meta protocol participants take a vote on a proposed modification of the object protocol rules. The participants of the meta protocol may initiate a meta-meta protocol to modify the rules of the meta protocol, or they may initiate a meta-meta-meta protocol to modify the rules of the meta-meta protocol, and so on. For simplicity, in this example *all* meta protocols are voting procedures (in other systems each meta protocol may be a different decision-making procedure). In general, in a k -level infrastructure, level 0 corresponds to the main (resource-sharing, in this example) protocol, while a protocol of level n , $0 < n \leq k-1$ (voting, in this example), is created, by the protocol participants of a level m , $0 \leq m < n$, in order to decide whether to modify the protocol rules of level $n-1$. The infrastructure for dynamic (resource-sharing) specifications is displayed in Figure 2.

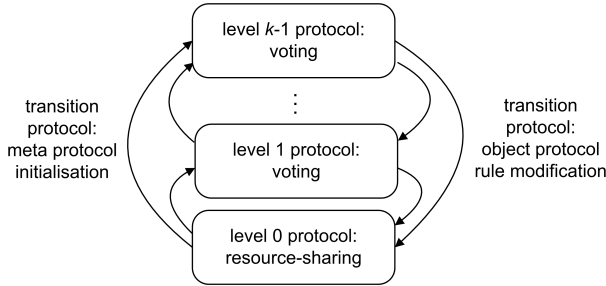


Fig. 2 A k -level Infrastructure for Dynamic Specifications.

Apart from object and meta protocols, the infrastructure for dynamic specifications includes ‘transition’ protocols — see Figure 2 — that is, procedures that express, among other things, the conditions in which an agent may successfully initiate a meta protocol (for instance, only the members of the board of directors may successfully

Table 3 A Subset of the Action Signature of D^{RS} (Part B).

Variable:	Domain:
PL	\mathbb{Z}^+
$SP, NSP, ASP, Motion$	a set of specification point ids
DoF_ID	a set of DoF ids
V	$\{for, against\}$
$Outcome$	$\{carried, not_carried\}$
Simple Fluent Constant:	Domain:
$role_of(Ag, PL)$	$\{subject, chair, fcs\}$
$dof(DoF_ID, SP)$	a set of DoF values
$actual_sp(PL)$	a set of specification point ids
$proposal(Ag, SP, PL), properties(SP, PL)$	Boolean
$threshold_d(PL), threshold_eu(PL),$	
$eu(SP, PL)$	\mathbb{Z}^+
Statically Determined Fluent Constant:	Domain:
$powPropose(Ag, SP, PL), powSecond(Ag, SP, PL),$	
$perPropose(Ag, SP, PL), oblPropose(Ag, SP, PL),$	
$powDeclare(Ag, Motion, Outcome, PL)$	Boolean
$distance(SP, SP, PL)$	\mathbb{Z}^+
Action Constant σ^{act} (Boolean):	
$propose(Ag, NSP, PL), second(Ag, NSP, PL),$	
$vote(Ag, V, PL), declare(Ag, Motion, Outcome, PL)$	

initiate a meta protocol in some organisations), the roles that each meta protocol participant will occupy, and the ways in which an object protocol is modified as a result of the meta protocol interactions. The components of the infrastructure for dynamic specifications, level 0 protocol, level n protocol ($n > 0$), and transition protocol, are discussed in Sections 5.1–5.4.

Table 3 shows a set of fluent and action constants of the action signature of D^{RS} that will be presented in the following sections. In order to distinguish between the protocol states of different protocol levels, we add a parameter, when necessary, in the representation of action and fluent constants, expressing the protocol level PL . For example, $role_of(Ag, PL)$ expresses the role Ag occupies in level PL . It is unnecessary to modify the syntax of action and fluent constants that are part of the specification of a single protocol level (such as the action constant $request_floor$ that concerns only level 0).

5.1 Level 0

For illustration purposes we chose a resource-sharing protocol — the specification of which was presented in Section 3 — as a level 0 protocol. A protocol specification consists of the ‘core’ rules that are always part of the specification, and the *Degrees of Freedom (DoF)*, that is, the specification components that may be modified at run-time. (A DoF can be seen, for example, as Vreeswijk’s ‘partial protocol specification’ [77].)

A protocol specification with l DoF creates an l -dimensional specification space where each dimension corresponds to a DoF. A point in the l -dimensional specification

space, or *specification point*, represents a complete protocol specification — a *specification instance* — and is denoted by an l -tuple where each element of the tuple expresses a ‘value’ of a DoF. Consider, for example, the resource sharing protocol with three DoF: the specification of the best candidate for the floor, the permission to assign the floor, and the permission to request a resource manipulation. The specification of these three protocol features may change at run-time — for instance, the best candidate may be determined randomly, on a first-come, first-served basis, priority may be given to subjects requesting a particular manipulation type, or to subjects that have not been sanctioned (these are possible values of the best candidate DoF). Regarding the second DoF, it may be forbidden to assign the floor to subjects that have been allocated the floor the last 3, 6, or 9 times. Finally, in this example, the holder may be permitted to request any type of resource manipulation from the Floor Control Server (FCS), or it may be permitted to request only the resource manipulation type expressed when it applied to the chair for the floor (these are possible values of the third DoF). In the resource-sharing example with these three DoF, a specification point is, for instance:

$$(fcs, 3, any_type)$$

According to the above specification point, the best candidate for the floor is determined on first-come, first-served basis (*fcs*), the maximum number of permitted consecutive allocations of the floor to a subject is 3, while the holder is permitted to request any type of resource manipulation (*any_type*).

In this example, the first DoF has 4 values, the second DoF has 3 values and the third DoF has 2 values. Consequently, the specification space contains $4 \times 3 \times 2 = 24$ specification points. In other examples we could have chosen different DoF (and/or DoF values), such as the specification of the permission and the obligation to request the floor, or perform any other protocol action. The classification of a specification component as a DoF is application-specific.

There are various reasons for which the agents may change at run-time the value of one or more DoF and thus the specification point. In the resource-sharing example, when the population of a system increases, the agents may decide to lower the limit of allowed consecutive allocations of the floor; when the number of agents violating the laws increases, it may be decided to give priority to agents that have not been sanctioned when computing the best candidate for the floor; etc. Furthermore, an agent may attempt to change at run-time the value of a DoF in order to satisfy its own private goals.

In any case, the value of one or more DoF, and thus the specification point, may change at run-time by means of a meta protocol. A discussion about meta protocols is presented in the following section.

We encode a protocol’s specification points in *C+* as follows:

$$\begin{aligned} \text{caused } dof(bc, sp_1) &= fcs \\ \text{caused } dof(per_assign, sp_1) &= 9 \\ \text{caused } dof(per_mpt, sp_1) &= any_type \end{aligned} \tag{9}$$

$$\begin{aligned} \text{caused } dof(bc, sp_2) &= rmt \\ \text{caused } dof(per_assign, sp_2) &= 9 \\ \text{caused } dof(per_mpt, sp_2) &= expressed_type \end{aligned} \tag{10}$$

The simple fluent constant *dof* records the value of a DoF — $dof(bc, sp_1) = fcs$, for example, denotes that, when the protocol’s specification point is sp_1 , the best candidate

(*bc*) for the floor is determined on a first-come, first-served basis. *per_assign* denotes the DoF concerning the permission to assign the floor while *per_mpt* denotes the DoF concerning the permission to request a resource manipulation. *rmt* is a value of the best candidate DoF, indicating that priority is given to subjects requesting a particular resource manipulation type. The above formalisation shows two example specification points: $sp_1 = (fcs, 9, any_type)$ and $sp_2 = (rmt, 9, expressed_type)$.

Each DoF value is defined by a set of rules — consider the following example formalisation:

$$\begin{aligned}
 &\text{caused } perRequestMpt(S, FCS, M) \text{ if} \\
 &\quad actual_sp(0) = ASP, \\
 &\quad dof(per_mpt, ASP) = any_type, \\
 &\quad role_of(FCS, 0) = fcs, \\
 &\quad holder(S)
 \end{aligned} \tag{11}$$

$$\begin{aligned}
 &\text{caused } perRequestMpt(S, FCS, M) \text{ if} \\
 &\quad actual_sp(0) = ASP, \\
 &\quad dof(per_mpt, ASP) = expressed_type, \\
 &\quad role_of(FCS, 0) = fcs, \\
 &\quad holder(S), \\
 &\quad requested(S) = M
 \end{aligned} \tag{12}$$

Rule (11) defines the *any_type* value of the DoF concerning the permission to request a resource manipulation type, while rule (12) defines the *expressed_type* value of this DoF. The *perRequestMpt* fluent constant expresses the permission to request a resource manipulation of a particular type (for instance, storing files of a particular type on a shared storage device, or executing applications of a particular type on a shared processor). According to rule (11), a subject *S* is permitted to request any resource manipulation type *M* if *S* is the holder of the resource. According to rule (12), the holder *S* of the resource is permitted to request only the manipulation type denoted when it applied to the chair for the floor (see the last condition of rule (12) — recall that the *requested* fluent constant records the subjects' requests for the floor). The simple fluent constant *actual_sp(PL)* records the actual specification point of protocol level *PL*. Rules (11) and (12) are replaceable in the sense that the participants of the resource-sharing protocol may active/deactivate one of them, at run-time, by changing the specification point in a way that the value of the DoF concerning the permission to request a resource manipulation type is modified. For example, moving from specification point sp_1 to sp_2 deactivates rule (11) and activates rule (12). The ways in which a specification point may change at run-time are presented next.

5.2 Level *n*

To provide a concrete example, we chose a three-level infrastructure for dynamic specifications. Moreover, both levels 1 and 2 are voting procedures, such as that presented in [61]. A presentation of a voting procedure specification is beyond the scope of this paper. Briefly, we assume a simple procedure including a set of voters casting their votes, 'for' or 'against' a particular motion, which would be in this example a proposed specification point change in level $n-1$, and a chair counting the votes and declaring the motion carried or not carried, based on the *standing rules* of the voting procedure — simple majority, two-thirds majority, etc.

Our infrastructure allows for the modification of the specification of all protocol levels apart from the top one. Consequently, we define DoF for all protocol levels apart from the top one. For the protocol of level 1 — voting — we chose to set as a DoF the standing rules of the voting procedure. In other words, a level 2 protocol may be initiated in order to decide whether level 1 voting should become, say, simple majority instead of two-thirds majority. Note that the voting procedures of levels 1 and 2 may not always have the same set of rules. For example, at a particular time-point level 2 voting may require a simple majority whereas level 1 voting may require a two-thirds majority (as mentioned above, the standing rules of level 1 constitute a DoF and thus the specification of this part of the protocol may change at run-time).

5.3 Transition Protocol

In order to change the specification point of level m , $m \geq 0$ (for example, to change the value of the best candidate DoF from *fcfs* to *rmt*), that is, in order to start a protocol of level $m+1$, the participants of level m need to follow a ‘transition’ protocol — see Figure 2. The infrastructure for dynamic specifications presented in this paper requires two types of transition protocol: one leading from the resource-sharing protocol to a voting one (level 0 to level 1 or 2), and one leading from one voting protocol to the other (level 1 to level 2). We will only present the first type of transition protocol; the latter type of protocol may be specified in a similar manner. An example transition protocol leading from resource-sharing to voting can be briefly described as follows. A subject S of the resource-sharing protocol proposes that the specification point of this level (or of level 1) changes. If S is empowered to make such a proposal, then the modification is directly accepted, without the execution of a meta protocol, provided that another subject exercises its power to second the proposal, and no other subject exercises its power to object to the proposal. If S is not empowered to make the proposal, or if the proposal is not seconded, then it is ignored. If the proposal is seconded, and there is an objection, then an argumentation procedure commences, the topic of which is the proposed specification point change, the proponent of the topic is S , the subject that made the proposal, and the opponent is the subject that objected to the proposal. The argumentation procedure is followed by a meta protocol (level 1 or 2) in which a vote is taken on the proposed specification point change.

In this example transition protocol we have specified the power to propose a specification point change as follows:

$$\begin{aligned}
 &\text{caused } \textit{powPropose}(Ag, NSP, PL) \text{ if} \\
 &\quad \textit{role_of}(Ag, 0) = \textit{subject}, \\
 &\quad \textit{actual_sp}(PL) = ASP, \text{ } ASP \neq NSP, \\
 &\quad \textit{protocol}(PL+1) = \textit{idle}, \\
 &\quad \textit{properties}(NSP, PL)
 \end{aligned} \tag{13}$$

An agent Ag is empowered to propose that the specification point of protocol level PL becomes NSP if the following conditions are satisfied. First, Ag occupies the role of subject in level 0. In this example, the chair of the resource-sharing protocol (level 0) is not empowered to propose a change of the specification point. Second, the actual specification point, ASP , is different from NSP . Third, there is no protocol taking place in level $PL+1$. A *protocol*(PL) simple fluent constant records whether a protocol of level PL is idle or executing. A protocol for changing the specification point of level

PL , that is, a protocol of level $PL+1$, may commence only if there is no other protocol of level $PL+1$ taking place.

Fourth, the specification instance corresponding to specification point NSP of level PL satisfies a set of properties — *properties* is a simple fluent constant. The properties that a specification instance should satisfy are application-specific. We may require, for example, that an agent is never forbidden and obliged to perform the same action, a floor control mechanism is ‘safe’ and ‘fair’ [23], and so on. In this example, we have chosen to specify that an agent is not empowered to propose the adoption of a specification point of the form $(rmt, *, any_type)$ ($*$ denotes any value of the second DoF), since these points correspond to specification instances that are, in some sense, ‘inconsistent’: priority for access to the resource is given to subjects with an expressed request of resource manipulation M (the value of the best candidate DoF is rmt), while it is permitted to actually request from FCS a different type of manipulation M' (the value of the DoF concerning the permission for actual resource manipulation is *any_type*).

CCALC is an automated reasoning tool allowing for proving protocol properties — recall that in Section 4 we proved properties of a specification instance of the resource-sharing protocol by means of CCALC’s query computation. Assuming that a protocol’s specification points are known before the commencement of the protocol execution, we may determine at design-time, with the use of CCALC, whether the specification instance corresponding to each specification point of a protocol level satisfies a set of desirable properties. Accordingly, we may set the value of the *properties* fluent constant (which is part of the transition protocol specification), thus avoiding, at run-time, the (time-consuming) task of proving protocol properties.

In other examples the specification of the power to propose a specification point change could have different, or additional conditions further constraining how a protocol specification may change at run-time. For instance, it may be required that, for any protocol level, the specification point does not change ‘too often’, or there may be an upper limit on the number of specification point changes (proposed by an agent). In Section 5.4 we present a way of further constraining the process of specification point change. Other ways of achieving that, such as the ones described above, could have been formalised.

A proposal for specification point change needs to be seconded by another subject having the institutional power to second the proposal in order to be directly enacted, or initiate the argumentation procedure leading to voting. We chose to specify the power to second a proposal for specification point change as follows:

$$\begin{aligned} \text{caused } powSecond(Ag, NSP, PL) \text{ if} \\ \text{role_of}(Ag, 0) = \text{subject}, \\ \text{proposal}(Ag', NSP, PL), \\ Ag \neq Ag' \end{aligned} \tag{14}$$

Ag is empowered to second any proposal for specification point change made by some other Ag' . The *proposal* simple fluent constant records proposals made by empowered agents (subjects, in this example).

We have specified that any subject is empowered to object to a proposal for specification point change.

Exercising the power to object to a proposal for specification point change initiates an argumentation procedure, the topic of which is the proposed change. To save space

we do not present here a specification of an argumentation procedure; see [6] for an example formalisation of such a procedure.

The completion of the argumentation taking place in the context of a transition protocol initiates a meta protocol. The latter protocol is a voting procedure concerning a proposed specification point change. The agents participating in this procedure and the roles they occupy are determined by the transition protocol that results in the voting procedure. We chose to specify that the chair of the resource-sharing protocol becomes the chair of the voting procedure. Furthermore, the agents occupying the role of voter, thus having the power to vote, are the subjects that have not been sanctioned for exhibiting ‘anti-social’ behaviour, that is, performing forbidden actions or not complying with obligations:

$$\begin{aligned} \text{caused } \text{role_of}(Ag, PL) = \text{voter} \text{ if} \\ \text{role_of}(Ag, 0) = \text{subject}, \\ \text{protocol}(PL) = \text{executing}, \quad PL > 0, \\ \neg \text{sanctioned}(Ag) \end{aligned} \quad (15)$$

The value of a $\text{protocol}(PL)$ constant becomes *executing*, in the case where $PL > 0$, at the end of the argumentation of the transition protocol that led to level PL . We chose not to relativise the constant recording sanctions to a protocol level. Therefore, the simple fluent constant *sanctioned* records ‘anti-social’ behaviour exhibited at any protocol level. Depending on the employed treatment of sanctions, ‘anti-social’ behaviour may be permanently recorded, thus permanently depriving a subject of participating in a meta level, or it may be temporarily recorded, thus enabling subjects to participate in a meta level after a specified period has elapsed from the performance of forbidden actions or non-compliance with obligations.

Clearly, meta level role-assignment may be specified in other ways. For example, it may be the case that agents that recently joined the system are not admitted in the meta level, or that priority is given to subjects that have had the least time accessing the shared resources, etc. In general, meta level role-assignment can be as complex as required by the application under consideration.

The fact that an agent may successfully start a protocol of level $m+1$ by proposing a change of the specification point of level m , does not necessarily imply that the specification point of level m will be changed. It is only if the motion of level $m+1$ — which is the proposed specification point for level m — is carried, that the specification point of level m will be changed. Consider the following rule expressing the outcome of a voting procedure of level $m+1$, that takes place in order to change the specification point in level m to NSP :

$$\begin{aligned} \text{declare}(VC, \text{Motion}, \text{carried}, PL+1) \text{ causes } \text{actual_sp}(PL) = NSP \text{ if} \\ \text{powDeclare}(VC, \text{Motion}, \text{carried}, PL+1), \\ \text{Motion} = NSP \end{aligned} \quad (16)$$

Exercising the power to declare the motion carried in level $PL+1$ results in changing the specification point in level PL to NSP , provided that the motion concerned the adoption of NSP . If the chair of the voting procedure VC did not declare the motion carried, or was not empowered to make the declaration, then the specification point would not have changed in level PL . To save space we do not present here the specification of the power to make a declaration.

Exercising the power to declare the motion carried in a meta level m may have additional effects. For example, we may explicitly specify whether or not deactivating

a rule, by changing the specification point of protocol level $m-1$, results in removing the effects of the rule that were produced prior to the rule deactivation. A discussion about specification change operations is presented in the last section of the paper.

5.4 Constraining the Process of Run-time Specification Modification

As already mentioned, all protocol levels apart from the top one have DoF and, therefore, their specification may be modified at run-time. In this section we present ways of evaluating a proposal for specification point change. Moreover, we present ways of constraining the enactment of proposals that do not meet the evaluation criteria.

One way of evaluating a proposal for specification point change is by modelling a dynamic protocol specification as a *metric space* [13]. More precisely, we compute the ‘distance’ between the proposed specification point and the actual point. We constrain the process of specification point change by permitting proposals only if the proposed point is not too ‘far’ from/‘different’ to the actual point. The motivation for formalising such a constraint is to favour gradual changes of a system specification. In what follows we describe how we may compute the distance between two specification points, and the conditions in which a proposal for specification point change is considered permitted.

We may follow two steps to compute the distance between two specification points sp and sp' , each represented as an l -tuple, where each element of the tuple expresses a DoF value. First, we may transform sp and sp' to l -tuples of non-negative integers qsp and qsp' respectively. To achieve that we can define an application-specific function v , that ‘ranks’ each DoF value, that is, associates every DoF value with a non-negative integer. The i -th element of qsp , qsp_i , has the value of $v(sp_i)$, where sp_i is the i -th element of sp (respectively, the i -th element of qsp' , qsp'_i , has the value of $v(sp'_i)$, where sp'_i is the i -th element of sp'). Second, we may employ a *metric* (or *distance function*), such as the Euclidean metric, to compute the distance between qsp and qsp' (the choice of a metric is application-specific — see [13] for a list of metrics). Depending on the employed metric, we may add weights on the DoF — for instance, we may require that the computation of the distance between qsp and qsp' is primarily based on the best candidate DoF rather than the other two DoF. The distance between sp and sp' is the distance between qsp and qsp' .

Alternatively, we may compute the distance between two specification points by defining an application-specific metric that does not necessarily rely on a quantification DoF values.

We may constrain run-time specification point change as follows:

$$\begin{aligned} \text{caused } \text{perPropose}(Ag, NSP, PL) \text{ iff} \\ \text{powPropose}(Ag, NSP, PL), \\ \text{actual_sp}(PL) = ASP, \\ \text{distance}(NSP, ASP, PL) \leq \text{threshold_d}(PL) \end{aligned} \quad (17)$$

$\text{perPropose}(Ag, NSP, PL)$ is a statically determined fluent constant denoting whether agent Ag is permitted to propose that the specification point of level PL becomes NSP . distance is a statically determined fluent constant computing the distance between any two specification points of a protocol level. The definition of distance includes a $C+$ formalisation of a chosen metric. $\text{threshold_d}(PL)$ is a simple fluent constant recording the maximum distance that a proposed specification point should have from the actual point in level PL . Different protocol levels may have different threshold_d values and

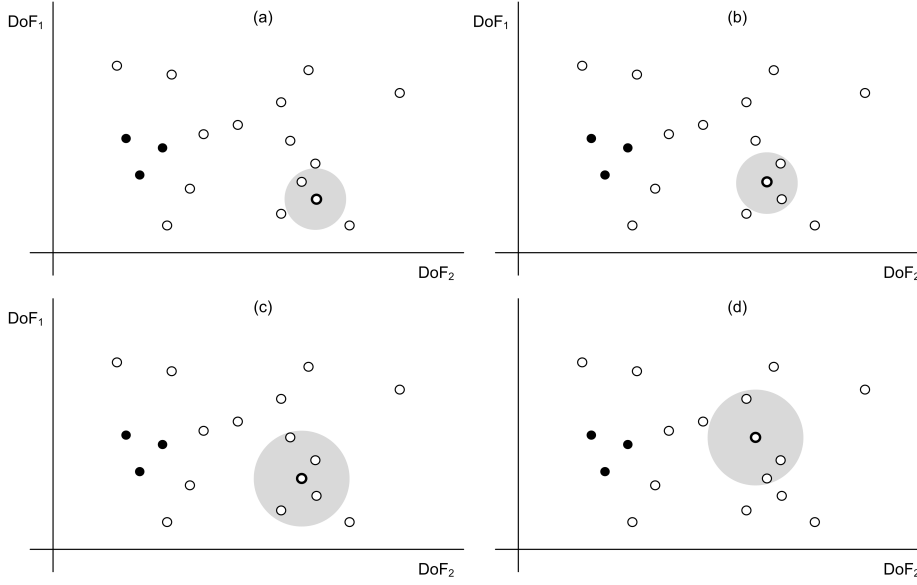


Fig. 3 Run-Time Change of Actual Specification Point and Maximum Allowed Distance between the Actual Specification Point and a Proposed Specification Point.

different metrics. According to rule (17), an agent Ag is permitted to propose that the specification point of level PL becomes NSP , if and only if Ag is empowered to make this proposal, and the distance between NSP and the actual specification point of level PL , ASP , is less or equal to a specified threshold.

Note that the maximum allowed distance between the actual point and a proposed point may change during the system execution. Consider, for example, snapshots (a)–(d) of the Euclidean specification space with two DoF shown in Figure 3. Black circles denote specification points representing specification instances that do not satisfy a set of protocol properties — consequently, an agent is never empowered to propose that the actual specification point becomes one of those points (see Section 5.3). White circles, on the other hand, denote specification points representing specification instances that satisfy the required protocol properties. The circle with the thick line denotes the actual specification point. The grey area denotes the maximum allowed distance between the actual specification point and a proposed point — this is expressed by $threshold_d$. Initially — snapshot (a) — only one specification point, sp , is within the grey area, that is, it is permitted, under certain circumstances, to move only to sp . Then — snapshot (b) — the actual specification point moves to sp . At this time, two specification points are within the grey area. Following this specification point change, the value of $threshold_d$ increases — snapshot (c) — that is, the size of the grey area increases, offering more options for permitted specification point change. In some systems, designated agents, such as ‘institutional agents’ [10], may increase, temporarily perhaps, the value of $threshold_d$ in order to allow for a greater system specification change, possibly as a result of sensing a substantial change of environmental, social, or other conditions. In other systems, changing the value of $threshold_d$ may be realised in a manner similar to that used for changing the actual specification point. Snapshot

(d) shows that the actual specification point moves to a point that could not have been directly reached had the value of *threshold_d* not increased, assuming that agents abide by the rules governing specification point change. Note that, in general, the members of a system may adopt *any* specification point that satisfies certain protocol properties — in the present example the agents may adopt any point depicted as a white circle. Moving outside of the grey area, however, is forbidden and the agent that proposed such a specification point change may be subject to penalty.

The designers of a system may further constrain the process of run-time specification modification by permitting a proposal for specification point change only if the expected system utility associated with the proposed point is ‘acceptable’ (in a sense to be specified below). Consider the following formalisation:

$$\begin{aligned}
 &\text{caused } \textit{perPropose}(Ag, NSP, PL) \text{ if} \\
 &\quad \textit{powPropose}(Ag, NSP, PL), \\
 &\quad \textit{actual_sp}(PL) = ASP, \\
 &\quad \textit{distance}(NSP, ASP, PL) \leq \textit{threshold_d}(PL), \\
 &\quad \textit{eu}(NSP, PL) \geq \textit{threshold_eu}(PL)
 \end{aligned} \tag{18}$$

$$\begin{aligned}
 &\text{caused } \textit{perPropose}(Ag, NSP, PL) \text{ if} \\
 &\quad \textit{powPropose}(Ag, NSP, PL), \\
 &\quad \textit{actual_sp}(PL) = ASP, \\
 &\quad \textit{distance}(NSP, ASP, PL) \leq \textit{threshold_d}(PL), \\
 &\quad \textit{eu}(NSP, PL) > \textit{eu}(ASP, PL)
 \end{aligned} \tag{19}$$

$$\text{default } \neg \textit{perPropose}(Ag, NSP, PL) \tag{20}$$

The simple fluent constant $\textit{eu}(NSP, PL)$ expresses the expected system utility associated with the specification point NSP of level PL , that is, the system utility expected to be achieved under the specification instance corresponding to NSP . The utility of a system may be defined in various ways; in a resource-sharing protocol, for example, the system utility may be defined in terms of the average time for servicing a request for the floor. The simple fluent constant $\textit{threshold_eu}(PL)$ expresses the minimum allowed expected system utility in level PL . The first three conditions of rules (18) and (19) are the same as the conditions of rule (17). According to rules (18)–(20), an agent Ag is permitted to propose that the specification point of level PL becomes NSP , if and only if Ag is empowered to make this proposal, NSP is not too ‘far’ from the actual point ASP , and

- the expected system utility associated with NSP exceeds a specified threshold, or
- the expected system utility associated with NSP is greater than the expected system utility associated with ASP .

Rules (18)–(20) are but one possible way to formalise the permission to propose a specification point change. We could have equally chosen to adopt, say, only rules (18) and (20), or rules (19) and (20).

Care should be taken when specifying the thresholds for distance and expected system utility. For example, it might be the case that, for certain values of these thresholds, it is never permitted to move from most specification points to any other point. CCALC is very useful in selecting the appropriate values for these thresholds. We may use CCALC to prove properties of a transition protocol under certain values of the distance and expected system utility thresholds, such as that, under certain circumstances it is permitted to move from some/all specification points to another

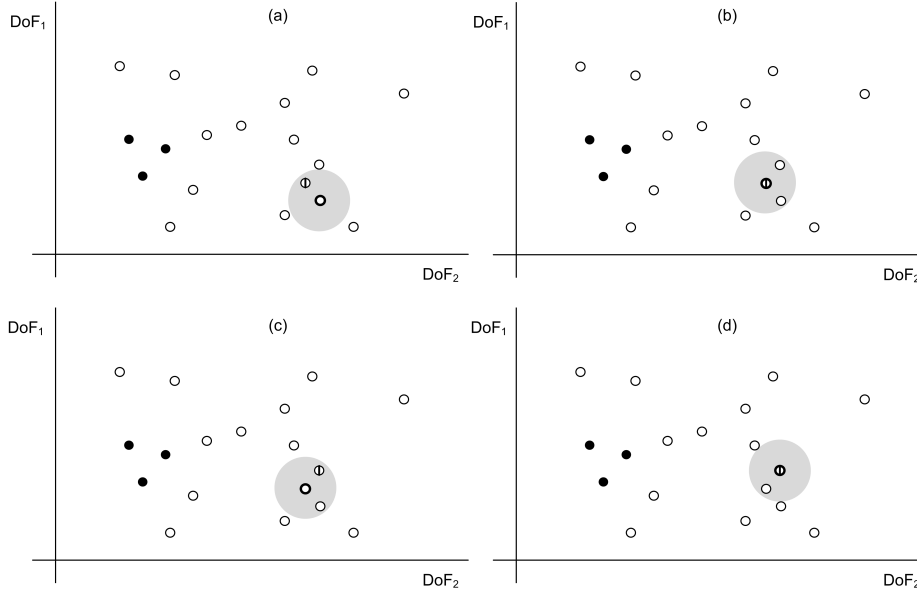


Fig. 4 Run-Time Change of Desired and Actual Specification Points.

point. In the next section we demonstrate the use of CCALC for proving properties of an infrastructure for dynamic protocol specification, including a transition protocol specification.

The system utility expected to be achieved under a specification instance depends on various conditions such as the size of the population of a system, the frequency of rule violation, and the available resources. Such conditions may fluctuate during a system's lifetime. Consequently, the system utility expected to be achieved under a specification instance may change over time. Figure 4 illustrates this issue; this figure shows four new snapshots of the Euclidean specification space presented in Figure 3. Recall that black circles denote specification points representing specification instances that do not satisfy a set of protocol properties, white circles denote specification points representing specification instances that do satisfy the required protocol properties, the circle with the thick line denotes the actual specification point, and the grey area denotes the maximum allowed distance between the actual specification point and a proposed point. The circle with the vertical line denotes the 'desired' specification point, that is, the specification point corresponding to the specification instance with the maximum expected system utility. To avoid clutter, we do not show the expected system utility of each specification point. The snapshots of Figure 3 show three specification point changes. First, the system members change the actual specification point in a way that it coincides with the desired point. Then, the desired specification point changes — such a change could be the result of a change in environmental conditions, for example. (In the present *C+* formalisation, we have written simple rules to compute the expected system utility associated with each specification point, and thus determine the desired point, under different environmental conditions. In other examples, institutional agents, or other types of agent may compute the expected system utility associated with each specification point, and therefore determine the desired point, under different

conditions.) Finally, the system members change once more the actual point in a way that it reaches the desired one. Recall that the members of a system may adopt any specification point (that satisfies certain protocol properties). In other words, it may not necessarily be the case that the members of a system try to reach the desired specification point, or move to a point that increases the expected system utility.

Apart from the desired point, the threshold concerning expected system utility (*threshold_eu*) may change over time, using institutional agents or some other means.

Similar to rules (18)–(20), we may express the permission to *second* a proposal for specification point change, or the *obligation to object* to such a proposal. Moreover, when the expected system utility associated with the actual specification point is below the specified threshold (*threshold_eu*), we could impose an *obligation to propose* a specification point change that, if accepted (the proposal), would change the actual point in a way that the associated expected system utility is increased. Below is a way of formalising such an obligation:

$$\begin{aligned} &\text{caused } \text{oblPropose}(Ag, NSP, PL) \text{ if} \\ &\quad \text{perPropose}(Ag, NSP, PL), \\ &\quad \text{actual_sp}(PL) = ASP, \\ &\quad \text{eu}(ASP, PL) < \text{threshold_eu}(PL) \end{aligned} \tag{21}$$

The statically determined fluent constant *oblPropose* expresses the obligation to propose a specification point change. According to the above rule, an agent *Ag* is obliged to propose that the specification point of level *PL* becomes *NSP* if *Ag* is permitted to propose the adoption of *NSP* (see rules (18)–(20) for an example definition of the permission to propose a specification point change), and the expected system utility associated with the actual specification point of level *PL*, *ASP*, is less than the *threshold_eu(PL)* value.

Note that *Ag*'s obligation to propose a specification point change may be terminated, even if *Ag* does not discharge it: a specification point with greater expected system utility may be adopted due to the proposal of some other agent. In this case the last condition of rule (21) may cease to hold and thus *Ag* will no longer be obliged to propose a specification point change.

6 Proving Properties of the Dynamic Resource-Sharing Protocol

Specifying a dynamic protocol in *C+* allows us to prove various properties of the specification. (Recall that in Section 4 we proved properties of a static resource-sharing protocol specification.) We may prove properties of the specification instances of level 0 and level *n*, *n* > 0, and the transition protocols. For example, we may prove that the transition protocols and level *n* protocols terminate within a fixed number of steps, that a level *n* protocol may be initiated at most a fixed number of times, and so on. Below we present a few example properties proven of the presented dynamic resource-sharing protocol, by means of CCALC query computation.

Recall that the *C+* action description D^{RS} , expressing the dynamic resource-sharing protocol, defines a three-level infrastructure; level 0 (resource-sharing) has three DoF, the specification of the best candidate for the floor, the permission to assign the floor, and the permission to request a resource manipulation. Level 1 voting has a single DoF, the standing rules of the voting procedure, while level 2 voting has no DoF. To conduct computational experiments, one has to make specific choices for

a set of parameters. For a concrete illustration we will present here experiments in which, for example, the distance between two specification points is computed with the use of a weighted Manhattan metric. Arbitrary values were chosen for the threshold distance between the actual specification point and a proposed point, as well as the threshold expected system utility. Moreover, the only specification points representing inconsistent, in some sense, specification instances are the level 0 points of the form $(rmt, *, any_type)$ (see Section 5.3). These specification points may not be reached because, in this example, no agent is empowered to propose the adoption of a specification point representing an inconsistent specification instance. Other choices concerning the experimental parameters could of course have been made, and the experiments repeated for those.

Property 6.1 *According to the specification instance corresponding to specification point sp_2 of level 0, there is no protocol state in which a subject is permitted to request a resource manipulation type different from that stated when applying for access to the resource.*

We instruct CCALC to compute all states s of D^{RS} such that

$$s \models actual_sp(0) = sp_2 \wedge perRequestMpt(S, FCS, M)$$

In other words, we are interested in computing all states in which the actual specification point of level 0 (resource-sharing) is sp_2 (see expression (10)), and a subject is permitted to request from the Floor Control Server (FCS) to manipulate the resource.

In all solutions computed by CCALC we have that

$$s \models perRequestMpt(S, FCS, requested(S))$$

Recall that the value of a simple fluent constant $requested(S)$ is the resource manipulation type (say, storing files of a particular type on the shared storage device) stated by subject S when it applied for access to the resource. According to the solutions produced by CCALC, a subject S is permitted to request from FCS only the type of resource manipulation expressed when S applied for access to the resource. This is due to $dof(per_mpt, sp_2) = expressed_type$ of expression (10) — recall that per_mpt represents the DoF concerning the permission to request a resource manipulation type — and rule (12).

Property 6.2 *There is no protocol state in which an agent is forbidden and obliged to propose a specification point change.*

We instruct CCALC to compute all states s of D^{RS} such that

$$s \models oblPropose(Ag, NSP, PL)$$

For all states s computed by CCALC we obtain

$$s \models perPropose(Ag, NSP, PL)$$

that is, there is no state in which an agent is obliged and forbidden to propose a specification point change. In the presented computational experiments, rules (18)–(20) express the conditions in which an agent is permitted to propose a specification

point change, while rule (21) expresses the conditions in which an agent is obliged to propose a specification point change.

Property 6.3 *Exercising the power to declare the outcome of the voting procedure of level n carried always changes the specification point of level $n-1$.*

We instruct CCALC to find all states s' such that

- (s, ε, s') is a transition of D^{RS} ,
- $s \models \text{powDeclare}(VC, NSP, \text{carried}, PL) \wedge \text{actual_sp}(PL-1) = ASP$, and
- $\varepsilon \models \text{declare}(VC, NSP, \text{carried}, PL)$.

For every state s' computed by CCALC we obtain

$$s' \models \text{actual_sp}(PL-1) = NSP$$

which denotes that the specification point of level PL has changed (in all solutions $ASP \neq NSP$). Rule (16) expresses the effects of exercising the power to declare the outcome of a voting procedure carried.

In some solutions to the above query, the subjects are obliged in the resulting state s' to propose a specification point change in level $PL-1$. This is due to the fact that, in these solutions, the expected system utility associated with the new specification point, NSP , is below the specified threshold (that is, $eu(NSP, PL-1) < \text{threshold_eu}(PL-1)$).

7 Animating the Dynamic Resource-Sharing Protocol

Apart from proving properties of a dynamic protocol specification, CCALC's query computation — in particular the computation of 'prediction' (temporal projection) queries, such as the queries used to prove Properties 4.2 and 6.3 — allows us to calculate, at run-time, the agents' powers, permissions, and obligations. Such information may be publicised to the members of a system, and may be provided by a central server or in various distributed configurations. (Further discussion of these architectural issues is outside the scope of this paper.) In this section we present an example execution (run) of the dynamic resource-sharing protocol, and the results obtained by CCALC's prediction query computation.

The narrative of events of the presented run is displayed in Table 4. The events of transition protocols, *propose*, *second*, *object*, level 1 and level 2 protocols, *vote* and *declare*, are indented. The last argument of a level 1 or 2 event indicates the protocol level in which the event took place. To save space, we group the votes that are in favour of (respectively, against) a motion. In the initial state of the presented run:

- The actual specification point of level 0 is $(fcs, 3, \text{any_type})$, that is, the best candidate for the floor is determined on a first-come, first-served basis, the maximum number of permitted consecutive allocations of the floor is 3, while the holder is permitted to request any type of resource manipulation (see rule (11)).
- The actual specification point of level 1 is (3_4m) , that is, a 75% majority is required (recall that level 1 has a single DoF).

Level 2 voting does not have a DoF. In these experiments level 2 voting requires a 75% majority. Details about the choices we made concerning the remaining experimental parameters were given in the previous section.

Table 4 Run of Dynamic Resource-Sharing Protocol.

Time	Action
0	<i>request_floor</i> (<i>sub</i> ₁ , <i>c</i> , <i>app</i> _A)
5	<i>request_floor</i> (<i>sub</i> ₂ , <i>c</i> , <i>app</i> _A)
6	<i>request_floor</i> (<i>sub</i> ₃ , <i>c</i> , <i>app</i> _A)
8	<i>request_floor</i> (<i>sub</i> ₅ , <i>c</i> , <i>app</i> _A)
14	<i>propose</i> (<i>sub</i> ₃ , <i>sp</i> ₂₆ , 1)
16	<i>object</i> (<i>sub</i> ₁ , <i>sp</i> ₂₆ , 1)
17	<i>second</i> (<i>sub</i> ₅ , <i>sp</i> ₂₆ , 1)
	transition protocol argumentation
28	<i>vote</i> ([<i>sub</i> ₂ , <i>sub</i> ₃ , <i>sub</i> ₄ , <i>sub</i> ₅ , <i>sub</i> ₆], <i>for</i> , 2)
30	<i>vote</i> (<i>sub</i> ₁ , <i>against</i> , 2)
31	<i>declare</i> (<i>c</i> , <i>sp</i> ₂₆ , <i>carried</i> , 2)
35	<i>propose</i> (<i>sub</i> ₅ , <i>sp</i> ₃ , 0)
36	<i>second</i> (<i>sub</i> ₃ , <i>sp</i> ₃ , 0)
39	<i>object</i> (<i>sub</i> ₂ , <i>sp</i> ₃ , 0)
	transition protocol argumentation
51	<i>vote</i> ([<i>sub</i> ₃ , <i>sub</i> ₄ , <i>sub</i> ₆], <i>for</i> , 1)
53	<i>vote</i> ([<i>sub</i> ₁ , <i>sub</i> ₂], <i>against</i> , 1)
54	<i>declare</i> (<i>c</i> , <i>sp</i> ₃ , <i>carried</i> , 1)
58	<i>assign_floor</i> (<i>c</i> , <i>sub</i> ₃)

The present example includes 7 agents, a chair *c* and 6 subjects *sub*₁–*sub*₆. In the beginning of the run-time activities *sub*₁, *sub*₂, *sub*₃ and *sub*₅ exercise their power to request the floor, all of them requiring to run applications of type *A* on the shared processor (see the third argument of *request_floor*). *sub*₃ and *sub*₅ aim to change the specification point of level 0 in a way that the value of the best candidate DoF becomes *random*, that is, the best candidate for the floor is chosen randomly from the list of subjects having pending floor requests. In this way *sub*₃ and *sub*₅ may acquire the floor faster. Before attempting to change the specification point of level 0, *sub*₃ and *sub*₅ attempt to change the specification point of level 1 in a way that level 1 voting requires simple majority as opposed to 75% majority. Therefore, fewer votes will be required in level 1 when *sub*₃ and *sub*₅ propose to change the specification point of level 0. The proposal for changing the specification point of level 1 takes place at time-point 14 — specification point *sp*₂₆ expresses that the standing rules require simple majority. At that time *sub*₃ is empowered to make the proposal (see rule (13)). Furthermore, *sub*₃ is permitted to exercise its power (see rules (18)–(20)) because: (i) the distance, in level 1, between the actual specification point (75% majority) and the proposed point *sp*₂₆ (simple majority) is less than the chosen threshold (*threshold_d*(1)), and (ii) the expected system utility associated with *sp*₂₆ is greater than the corresponding threshold (*threshold_eu*(1)). *sub*₃'s proposal is followed by an objection, a secondment, and an argumentation. Then level 2 voting commences; the motion is the adoption of *sp*₂₆ in level 1. *sub*₂–*sub*₆ vote for the motion while *sub*₁ votes against it. At time-point 31 the motion of level 2 is declared carried (recall that level 2 requires 75% majority) and thus the specification point of level 1 becomes *sp*₂₆ (see rule (16)), meaning that the standing rules of level 1 change to simple majority.

*sub*₅ proposes at time-point 35 the adoption of specification point *sp*₃=(*random*, 3, *any_type*) in level 0. *sub*₅ is empowered to make the proposal at that time. However, *sub*₅ is not permitted to exercise its power because the expected system utility of *sp*₃ is less than *threshold_eu*(0), and less than the expected system

utility associated with the actual point (*fcfs*, 3, *any_type*) of level 0. Consequently, *sub*₅ is sanctioned for performing a forbidden action and thus cannot participate in level 1 to vote (see rule (15)). *sub*₃, *sub*₄ and *sub*₆ vote for the motion of level 1 — the adoption of *sp*₃ in level 0 — while *sub*₁ and *sub*₂ vote against it. At time-point 54 the motion of level 1 is declared carried (recall that level 1 now requires a simple majority) and thus the specification point of level 0 becomes *sp*₃, meaning that the best candidate for the floor is chosen randomly from the list of subjects having pending floor requests. At time 58 *c* exercises its power (see rule (5)) to assign the floor to *sub*₃.

8 Summary, Related and Further Work

We presented an infrastructure for dynamic specifications for open MAS, that is, specifications that are developed at design-time but may be modified at run-time by the members of a system. Any protocol for open MAS may be in level 0 of our infrastructure, whereas any protocol for decision-making over specification change may be in level *n*, *n*>0. The level 0/level *n*/transition protocols can be as complex/simple as required by the application in question.

We employed *C+*, an action language with explicit transition system semantics to formalise dynamic specifications. Moreover, we employed CCALC, an automated reasoning tool for proving properties of the specifications and assimilating narratives of events. On the one hand, therefore, we may provide design-time services, proving properties, such as termination, of the various protocols of our infrastructure, and on the other hand, we may offer run-time services, calculating the system state current at each time, including the powers, permissions and obligations of the agents.

Chopra and Singh [15] have presented a way of adapting ‘commitment protocols’ [70–72, 74] according to context, or the preferences of agents in a given context. They formalise protocols and ‘transformers’, that is, additions/enhancements to an existing protocol specification that handle some aspect of context or preference. Depending on the context or preference, a protocol specification is complemented, at *design-time*, by the appropriate transformer thus resulting in a new specification. Unlike Chopra and Singh, we are concerned here with the *run-time* adaptation of a protocol specification and, therefore, we developed an infrastructure — meta protocols, transition protocols — to achieve that.

Several approaches have been proposed in the literature for run-time specification change of norm-governed MAS. Serban and Minsky [67], for example, have presented a framework for law change in the context of ‘Law-Governed Interaction’ (LGI) [55, 56, 58, 60, 78]. LGI is an abstract regulatory mechanism that satisfies the following principles: statefulness, that is, the regulatory mechanism is sensitive to the history of interaction between the regulated components, decentralisation, for scalability, and generality, that is, LGI is not biased to a particular type of law. LGI is an abstraction of a software mechanism called Moses [57, 59] which can be used to regulate distributed systems. Moses employs regimentation devices that monitor the behaviour of agents, block the performance of forbidden actions and enforce compliance with obligations.

It has been argued [46] that regimentation is rarely desirable (it results in a rigid system that may discourage agents from entering it [62]), and not always practical. In any case, violations may still occur even when regimenting a MAS (consider, for instance, a faulty regimentation device). For all of these reasons, we have to allow for non-compliance and sanctioning and not rely exclusively on regimentation mechanisms.

Serban and Minsky were concerned in [67] with architectural issues concerning law change. They presented a framework with which a law change is propagated to the distributed regimentation devices, taking into consideration the possibility that during a ‘convergence period’ various regimentation devices operate under different versions of a law, due to the difficulties of achieving synchronised, atomic law update in distributed systems.

There are several other approaches in the literature concerned with architectural issues of run-time specification change — [14, 24, 37] are but a few examples. These issues — various distributed configurations for computing the normative relations current at each time — are beyond the scope of this paper, and will be considered in future work.

Bou and colleagues [9, 10] have presented a mechanism for the run-time modification of the norms of an ‘electronic institution’ [25–29, 34, 64]. These researchers have proposed a ‘normative transition function’ that maps a set of norms (and goals) into a new set of norms: changing a norm requires changing its parameters, or its effect, or both. The ‘institutional agents’, representing the institution, are observing the members’ interactions in order to learn the normative transition function, so that they (the institutional agents) will directly enact the norms enabling the achievement of the ‘institutional goals’ in a given scenario. Unlike Bou and colleagues, we do not necessarily rely on designated agents to modify norms. We presented an infrastructure with which any agent may (attempt to) adapt the system specification. This does not exclude the possibility, however, that, in some applications, designated agents are given, under certain circumstances, the institutional power to directly modify the system specification.

Identifying *when* (to propose) to change a system specification during the system execution, as done in the work of Bou and colleagues with respect to the ‘institutional goals’, is a fundamental requirement for adaptive MAS. Addressing this requirement, however, is out of the scope of this paper.

Boella and colleagues [8] have developed a formal framework for representing norm change — see [12] for a recent survey on formal models of norm change. The framework of norm change presented in [8] is produced by replacing the propositional formulas of the Alchourrón, Gärdenfors, Makinson (AGM) framework of theory change [2] with *pairs* of propositional formulas — the latter representing norms — and adopting several principles from input/output logic [53]. The resulting framework includes a set of postulates defining norm change operations, such as norm contraction.

Governatori and colleagues [40–42] have presented variants of a Temporal Defeasible Logic [38, 39] to reason about different aspects of norm change. These researchers have represented meta norms describing norm modifications by referring to a variety of possible time-lines through which the elements of a norm-governed system, and the conclusions that follow from them, can or cannot persist over time. Governatori et al. have formalised norm change operations according to which norms are removed with all their effects, as well as operations according to which norms are removed but all or some of their effects propagate if obtained before the modification.

$C+$, along with proposed extensions [20] of this language, allows for the formalisation of relatively complex norm change operations. We expect that the expressiveness of $C+$ is adequate for representing norm change operations for a wide range of software MAS. The infrastructure presented in this paper, however, may be formalised using other languages. We chose $C+$ because it enables the formal representation of the (direct and indirect) effects of actions and default persistence (inertia) of facts, has a transition system semantics and thus there is a link to a wide range of other formalisms

and tools based on transition systems (later in this section we describe a way to exploit this link by combining *C+* with standard model checkers), and has direct routes to implementation.

Rubino and Sartor [65] have presented a taxonomy of ‘source norms’, that is, norms empowering the members of a system to modify a set of other norms. The so-called ‘agreement-based source norms’ may be viewed as the norms of a meta protocol in our infrastructure, in the sense that they allow for norm change based on the deliberation of a group of agents. Rubino and Sartor employ the logic of the PRATOR system for defeasible argumentation [63] to express source norms. The formalisation of the ideas presented in [65], however, is restricted; consequently, this line of work cannot be used yet to support dynamic MAS.

Run-time specification change has long been studied in the field of argumentation. Loui [51], for example, identified the need to allow for the run-time modification of argumentation protocol rules by means of meta argumentation. Vreeswijk [77] also investigated forms of meta argumentation. The starting point for this work was two basic observations. Firstly, that there are different protocols appropriate for different contexts (for example, quick and shallow reasoning when time is a constraint; restricted number of counter-arguments when there are many rules and cases; etc). Secondly, that ‘points of order’, by which a participant may steer the protocol to a desired direction, are standard practice in dispute resolution meetings. Vreeswijk then defined a formal protocol for disputes in which points of order can be raised to allow (partial) protocol changes to be debated. A successful ‘defence’ meant that the parties in the dispute agreed to adopt a change in the protocol, and the rules of dispute were correspondingly changed.

As already mentioned, our work is motivated by Brewka’s dynamic argument systems [11]. Like Vreeswijk’s work, these are argument systems in which the protagonists of a disputation may start a meta level debate, that is, the rules of order become the current point of discussion, with the intention of altering these rules. Unlike Vreeswijk’s work, there may be more than one meta argumentation level.

A key difference between our work and Brewka’s approach, and more generally, a key difference between our work and related research, including all approaches discussed in this section, is that we formalise the transition protocol leading from an object protocol to a meta protocol. More precisely, we distinguish between successful and unsuccessful attempts to initiate a meta protocol (exercising the institutional power to propose/second/object to a specification change vs proposing/seconding/objecting to a specification change without the necessary power), evaluate proposals for specification change by modelling a specification as a metric space, and by taking into consideration the effects of accepting a proposal on system utility, constrain the enactment of proposals that do not meet the evaluation criteria, and formalise procedures for role-assignment in a meta level.

In this section we focused on the facilities offered by related approaches with respect to system specification *change*. A comparison of our work with commitment protocols (including [17, 31–33, 75]), LGI, electronic institutions, Brewka’s argument systems, as well as other approaches for specifying norm-governed systems, from the viewpoint of *static* specifications, may be found in [5, 6].

We have been concerned with a particular aspect of ‘organised adaptation’ [76]: the run-time modification of the ‘rules of the game’ of norm-governed systems. Clearly there are other aspects of (organised) adaptation such as the run-time alteration of the (trading and other) relationships between agents, the assignment of roles to agents,

and the goals of a system. [18, 21, 22, 30, 43–45, 48, 54, 69, 73] are but a few examples of studies of adaptive systems.

We employed CCALC for the provision of run-time services — for instance, to compute the system state current at each time. CCALC, however, can become inefficient when considering action descriptions defining large transition systems. (CCALC is not the only means by which *C+* action descriptions can be executed. In [35] it is shown how *C+* action descriptions can be translated into the formalism of (extended) logic programs.) We have also used versions of the Event Calculus (EC) [49] to specify and execute dynamic specifications [3]. EC is a simple and flexible formalism that is efficiently implemented for narrative assimilation. Our Prolog EC implementation, however, does not offer facilities for proving properties of a specification or planning. More importantly, we also lose the explicit transition system semantics which we see as a very important advantage of the *C+* formulation. A discussion comparing the use of EC and *C+* for developing executable MAS specifications can be found in [7].

A direction for further work is to employ a single formalism for efficient execution (narrative assimilation and proving properties) of (dynamic) MAS specifications. Some first steps are reported by Craven [19] who investigates methods for efficient EC-like query evaluation for (a subset of) the *C+* language, and for integrating action descriptions in this language with standard model checking systems (specifically NuSMV [16]). Norm-governed system properties expressed in temporal logics such as Computation Tree Logic can then be verified by means of standard model checking techniques on a transition system defined using the *C+* language.

Acknowledgements

This paper is a significantly updated and extended version of [3, 4]. We would like to thank the reviewers and participants of the Eighth International Conference on Autonomous Agents and Multi-Agent Systems and the Twelfth International Conference on Artificial Intelligence & Law, who gave us useful feedback. We would also like to thank Lloyd Kamara for his comments on various drafts of the paper.

References

1. V. Akman, S. Erdogan, J. Lee, V. Lifschitz, and H. Turner. Representing the zoo world and the traffic world in the language of the Causal Calculator. *Artificial Intelligence*, 153(1–2):105–140, 2004.
2. C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.
3. A. Artikis. Dynamic protocols for open agent systems. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 97–104. ACM, 2009.
4. A. Artikis. Formalising dynamic protocols for open agent systems. In *Proceedings of International Conference on Artificial Intelligence & Law (ICAAIL)*, pages 68–77. ACM, 2009.
5. A. Artikis and M. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.
6. A. Artikis, M. Sergot, and J. Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15):776–804, 2007.
7. A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1), 2009.

8. G. Boella, G. Pigozzi, and L. van der Torre. Normative framework for normative system change. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 169–176. ACM Press, 2009.
9. E. Bou, M. López-Sánchez, and J. Rodríguez-Aguilar. Towards self-configuration in autonomic electronic institutions. In *Proceedings of Workshop on Coordination, Organization, Institutions and Norms in agent systems*, LNCS 4386, pages 220–235. Springer, 2007.
10. E. Bou, M. López-Sánchez, and J. Rodríguez-Aguilar. Using case-based reasoning in autonomic electronic institutions. In *Proceedings of Workshop on Coordination, Organization, Institutions and Norms in agent systems*, LNCS 4870, pages 125–138. Springer, 2008.
11. G. Brewka. Dynamic argument systems: a formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation*, 11(2):257–282, 2001.
12. J. Broersen. Issues in designing logical models for norm change. In G. Vouros, A. Artikis, K. Stathis, and J. Pitt, editors, *Proceedings of International Workshop in Organised Adaptation on Multi-Agent Systems (OAMAS)*, volume LNCS 5368, pages 1–17. Springer, 2009.
13. V. Bryant. *Metric Spaces*. Cambridge University Press, 1985.
14. R. Chadha, H. Cheng, Y.-H. Cheng, J. Chiang, A. Ghetie, G. Levin, and H. Tanna. Policy-based mobile ad hoc network management. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 35–44. IEEE Computer Society, 2004.
15. A. Chopra and M. Singh. Contextualizing commitment protocols. In *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1345–1352. ACM, 2006.
16. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of International Conference on Computer-Aided Verification (CAV 2002)*, Copenhagen, July 2002, LNCS 2404. Springer, 2002. See <http://nusmv.iirst.itc.it>.
17. M. Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of Workshop on Agent Languages and Communication Policies*, pages 21–29, 2000.
18. A. Rocha Costa and G. Pereira Dimuro. A minimal dynamical MAS organisation model. In V. Dignum, editor, *Multi-Agent Systems — Semantics and Dynamics of Organisational Models*. IGI Global, 2009.
19. R. Craven. *Execution Mechanisms for the Action Language C+*. PhD thesis, University of London, September 2006.
20. R. Craven and M. Sergot. Distant causation in C+. *Studia Logica*, 79(1):73–96, 2005.
21. S. DeLoach. OMACS: A framework for adaptive, complex systems. In V. Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organisational Models*. IGI Global, 2009.
22. S. DeLoach, W. Oyenon, and E. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2008.
23. H.-P. Dommel and J. J. Garcia-Luna-Aceves. Floor control for multimedia conferencing and collaboration. *Multimedia Systems*, 5(1):23–38, 1997.
24. N. Dulay, E. Lupu, M. Sloman, and N. Damianou. A policy deployment model for the Ponder language. In *Proceedings of International Symposium on Integrated Network Management*, pages 14–18. IEEE/IFIP, 2001.
25. M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In C. Castelfranchi and L. Johnson, editors, *Proceedings of Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1045–1052. ACM Press, 2002.
26. M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In J.-J. Meyer and M. Tambe, editors, *Intelligent Agents VIII: Agent Theories, Architectures, and Languages*, LNAI 2333, pages 348–366. Springer, 2002.
27. M. Esteva, J. Rodríguez-Aguilar, J. Arcos, C. Sierra, and P. Garcia. Institutionalising open multi-agent systems. In E. Durfee, editor, *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*, pages 381–382. IEEE Press, 2000.
28. M. Esteva, J. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. Arcos. On the formal specifications of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce*, LNAI 1991, pages 126–147. Springer, 2001.
29. M. Esteva, J. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. Verifying norm consistency in electronic institutions. In *Proceedings of the AAAI-04 Workshop on Agent Organizations: Theory and Practice*, pages 8–14, 2004.

30. C.B. Excelente-Toledo and N.R. Jennings. The dynamic selection of coordination mechanisms. *Autonomous Agents and Multi-Agent Systems*, 9:55–85, 2004.
31. N. Fornara and M. Colombetti. *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter Formal specification of artificial institutions using the event calculus. IGI Global, 2009.
32. N. Fornara, F. Viganò, and M. Colombetti. Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, 14:121–142, 2007.
33. N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti. Artificial institutions: A model of institutional reality for open multiagent systems. *Artificial Intelligence and Law*, 16:89–105, 2008.
34. A. García-Camino, P. Noriega, and J. Rodríguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 667–673. ACM Press, 2005.
35. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.
36. E. Giunchiglia, J. Lee, V. Lifschitz, and H. Turner. Causal laws and multi-valued fluents. In *Proceedings of Workshop on Nonmonotonic Reasoning, Action and Change (NRAC)*, 2001.
37. S. Godic and T. Moses. Oasis extensible access control markup language (xacml), version 2.0. <http://www.oasis-open.org/committees/xacml/index.shtml>, March 2005.
38. G. Governatori, V. Padmanabhab, and A. Rotolo. Rule-based agents in temporalised defeasible logic. In *Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, LNCS 4099, pages 31–40. Springer, 2006.
39. G. Governatori, M. Palmirani, R. Riveret, A. Rotolo, and G. Sartor. Norm modifications in defeasible logic. In *Proceedings of Conference on Legal Knowledge and Information Systems (JURIX)*, pages 13–22. IOS Press, 2005.
40. G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment. Part I: Revision of defeasible theories. In R. van der Meyden and L. van der Torre, editors, *Proceedings of Conference on Deontic Logic in Computer Science (DEON)*, LNCS 5076, pages 3–18. Springer, 2008.
41. G. Governatori and A. Rotolo. Changing legal systems: Abrogation and annulment. Part II: Temporalised defeasible logic. In G. Boella, G. Pigozzi, M. Singh, and H. Verhagen, editors, *Proceedings of Workshop on Normative Multiagent Systems (NORMAS)*, pages 112–127, 2008.
42. G. Governatori, A. Rotolo, R. Riveret, M. Palmirani, and G. Sartor. Variants of temporal defeasible logics for modelling norm modifications. In *Proceedings of International Conference on Artificial Intelligence & Law (ICAIL)*. ACM, 2007.
43. M. Hoogendoorn. Adaptation of organizational models for multi-agent systems based on max flow networks. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1321–1326, 2007.
44. M. Hoogendoorn, C. Jonker, M. Schut, and J. Treur. Modeling centralized organization of organizational change. *Computational & Mathematical Organization Theory*, 13(2):147–184, 2007.
45. J. F. Hübner, J. S. Sichman, and O. Boissier. Using the MOISE+ for a cooperative framework of MAS reorganisation. In *Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA)*, volume LNAI 3171, pages 506–515. Springer, 2004.
46. A. Jones and M. Sergot. On the characterisation of law and computer systems: the normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. J. Wiley and Sons, 1993.
47. A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
48. R. Kota, N. Gibbins, and N. Jennings. Self-organising agent organisations. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 797–804. ACM, 2009.
49. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.
50. J. Lee, V. Lifschitz, and H. Turner. A representation of the zoo world in the language of the Causal Calculator. In *Proceedings of Symposium on Formalizations of Commonsense Knowledge*, 2001.
51. R. Loui. Process and policy: Resource-bounded non-demonstrative argument. Technical report, Washington University, Department of Computer Science, 1992.

52. D. Makinson. On the formal representation of rights relations. *Journal of Philosophical Logic*, 15:403–425, 1986.
53. D. Makinson and L. van der Torre. Input-output logics. *Journal of Philosophical Logic*, 29:383–408, 2000.
54. C. Martin and K. S. Barber. Adaptive decision-making frameworks for dynamic multi-agent organizational change. *Autonomous Agents and Multi-Agent Systems*, 13(3):391–428, 2006.
55. N. Minsky. The imposition of protocols over open distributed systems. *IEEE Transactions on Software Engineering*, 17(2):183–195, 1991.
56. N. Minsky. On conditions for self-healing in distributed software systems. In *International Workshop on Active Middleware Services*. IEEE Computer Society, 2003.
57. N. Minsky. *Law-Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction and a Reference Manual)*, 2005. Retrieved October 24, 2008, from <http://www.moses.rutgers.edu/documentation/manual.pdf>.
58. N. Minsky. Decentralised regulation of distributed systems: Beyond access control. Submitted for publication. Retrieved October 24, 2008, from <http://www.cs.rutgers.edu/~minsky/papers/IC.pdf>, 2008.
59. N. Minsky and T. Murata. On manageability and robustness of open multi-agent systems. In *Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications*, LNCS 2940, pages 189–206. Springer, 2004.
60. N. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(3):273–305, 2000.
61. J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in multi-agent systems. *Computer Journal*, 49(2):156–170, 2006.
62. H. Prakken. Formalising Robert’s rules of order. Technical Report 12, GMD – German National Research Center for Information Technology, 1998.
63. H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7(1):25–75, 1997.
64. J. Rodriguez-Aguilar and C. Sierra. Enabling open agent institutions. In K. Dautenhahn, A. Bond, L. Canamero, and B. Edmonds, editors, *Socially Intelligent Agents: Creating relationships with computers and robots*, pages 259–266. Kluwer Academic Publishers, 2002.
65. R. Rubino and G. Sartor. Source norms and self-regulated institutions. In P. Casanovas, G. Sartor, N. Casellas, and R. Rubino, editors, *Computable Models of the Law, Languages, Dialogues, Games, Ontologies*, LNCS 4884. Springer, 2008.
66. J. Searle. What is a speech act? In A. Martinich, editor, *Philosophy of Language*, pages 130–140. Oxford University Press, third edition, 1996.
67. C. Serban and N. Minsky. In vivo evolution of policies that govern a distributed system. In *International Symposium on Policies for Distributed Systems and Networks*, pages 134–141. IEEE, 2009.
68. M. Sergot. $(C+)^{++}$: An action language for modelling norms and institutions. Technical Report 2004/8, Department of Computing, Imperial College London, 2004.
69. Y. Shoham and M. Tennenholtz. On the emergence of social conventions: modeling, analysis and simulations. *Artificial Intelligence*, 94(1-2):139–166, 1997.
70. M. Singh. Agent communication languages: rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
71. M. Singh. An ontology for commitments in multiagent systems: towards a unification of normative concepts. *Artificial Intelligence and Law*, 7(1):97–113, 1999.
72. M. Singh. A social semantics for agent communication languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, LNCS 1916, pages 31–45. Springer, 2000.
73. G. Tesauro, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White. A multi-agent systems approach to autonomic computing. In *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 464–471. ACM, 2004.
74. M. Venkatraman and M. Singh. Verifying compliance with commitment protocols. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.
75. F. Viganò and M. Colombetti. Symbolic model checking of institutions. In *Proceedings of Conference on Electronic commerce*, pages 35–44. ACM, 2007.

- 76. G. Vouros, A. Artikis, K. Stathis, and J. Pitt, editors. *Organized Adaption in Multi-Agent Systems*, volume LNAI 5368. Springer, 2008.
- 77. G. Vreeswijk. Representation of formal dispute with a standing order. *Artificial Intelligence and Law*, 8(2/3):205–231, 2000.
- 78. W. Zhang, C. Serban, and N. Minsky. Establishing global properties of multi-agent systems via local laws. In D. Weyns, editor, *Environments for Multiagent Systems III*, volume LNAI 4389. Springer, 2007.